An Implementation Model of Policy Based DiffServ Quality of Service Management System

Si-Ho Cha, Dong-Ho Lee, Kuk-Hyun Cho

Dept. of Computer Science, Kwangwoon University
447-1, Wolgye-Dong, Nowon-Gu, Seoul, 139-701, Korea
{sihoc, dhlee, khcho}@cs.kwangwoon.ac.kr

Abstract. DiffServ is a technique to provide QoS in an efficient and scalable way. However, DiffServ model does not provide a complete QoS management framework and its implementation model. This paper proposes an implementation model that supports DiffServ policies for managing QoS of DiffServ networks. The implementation model is conform to MVC model, and based on IETF policy framework and EJB technologies. DiffServ QoS policies are represented as valid XML files and translated to low-level EJB policy beans in the EJB container of the DiffServ QoS management system. The low-level policy beans consist of Classifier policy, Meter & Action policy, and Queuing & Scheduling policy. These Policies are distributed using SNMP to control and monitor QoS of DiffServ networks. To support this work, we implemented the SNMP DiffServ agent containing DiffServ MIB on the Linux-based DiffServ routers. The proposed implementation model is on developing in our laboratory.

1 Introduction

Today’s IP networks are translating from best-effort service model to one that can provide different service levels for specific Quality of Service (QoS) requirements. In best-effort service model, the traffic is processed as quickly as possible, but there are no guarantees for QoS. To solve this problem, the Internet Engineering Task Force (IETF) proposed two models: Integrated Services (IntServ) and Differentiated Services (DiffServ). The main disadvantage of the IntServ is that the required per-flow state information and QoS treatment in the core IP network pose severe scalability problems [1]. The DiffServ model supports aggregate traffic classes rather than individual flows and provides different QoS to different classes of packet in IP networks. However, the DiffServ model does not provide a complete QoS management framework and any implementation models. It is not possible to provide QoS without the network and service management support.

To solve this problem, this paper describes the implementation model of a policy-based QoS management system for DiffServ networks. Policy-based QoS management means to manage the quality of service of a network through high-level business rules or directives. Our implementation model is based on the IETF policy framework and supports DiffServ policies for QoS management of DiffServ routers. In the
implementation model, DiffServ QoS policies are represented as valid XML files and are translated as Enterprise JavaBeans (EJB) components in the DiffServ QoS management system based on EJB technology. And the policy distribution is processed by Simple Network Management Protocol (SNMP).

The rest of the paper is organized as follows. Section 2 presents an overview of DiffServ QoS and policy-based management. In section 3, we describe our implementation model of policy-based DiffServ QoS management system. Section 4 presents the implementation of the SNMP agent and the DiffServ MIB on the DiffServ routers, as well as the implementation model described in section 3. Finally in section 5 we present a summary and point to our future work.

2 DiffServ QoS and Policy Based Management

2.1 DiffServ QoS

DiffServ provides service differentiation for aggregate IP packet streams by implementing different Per Hop Behaviors (PHBs) for different Differentiated Service Code Point (DSCP) values. The PHB is the DiffServ behavior expected at each node, namely, the treatment afforded to packets at each DiffServ router. The DSCP is simply the value contained within the first 6 bits of the Differentiated Services (DS) byte in the IP header. It indicates a particular PHB that should be given to the corresponding packet [2]. Currently, four types of PHBs are specified for use within the DiffServ network: default behavior (best-effort), class selector (CLS), expedited forwarding (EF), and assured forwarding (AF).

There are two kinds of DiffServ routers, one is edge router located at the boundary of the DiffServ domain and the other is core router interconnecting edgy routers. The edge router marks IP packets with the DSCP and the core router forwards the packets according to a proper PHB based on the DSCP in the IP header. Traffic entering a DiffServ network through an edgy router is first classified, then passed through some form of admission filter, intended to shape the traffic to meet the policy requirements associated with the classification. The shaped traffic stream is then assigned to a particular behavior aggregate, by marking IP packets with the DSCP. When the stream is passed through the DiffServ network, this DSCP value triggers a selected PHB from all core routers of the network. This procedure leads to the provision of QoS to networks and needs the management support to provide QoS.

2.2 Policy Based Management

Policy-based management means to manage the behavior of a network through high-level business rules or directives. One of the key motivations behind policy-based management is flexibility and graceful evolution of the management system so that it can adapt to changing requirements over a long period of time. This is achieved by
disabling, modifying old policies and by introducing new ones in order to meet changing requirements [3].

Figure 1 presents the IETF policy framework [4] and network management architecture. The Policy Management Tool (PMT) takes as input the high-level policies that a user or administrator enters in the network and converts them to a much more detailed and precise low-level policy description that can apply to the various devices in the network. The Policy Repository is used to store the policies generated by the PMT. The PR can be used to store both high-level and low-level policies. The Policy Decision Point (PDP) is responsible for interpreting the policies stored in the PR and communicating them to the Policy Enforcement Point (PEP). The devices that can apply and execute the different policies are known as PEPs. In the network management, a PMT and PDP may be corresponded to a user interface and management system (manager), while PEP corresponds to a managed node.

![Fig. 1. IETF Policy Architecture and Network Management Architecture](image)

A DiffServ Network consists of two types of PEPs: edge routers and core routers. Edge routers must know the business rules (policies) which determine how different packets should be marked into the DS field of the IP header. The edge routers also can implement various types of rate control, limiting the amount of network bandwidth to be used by a particular type of traffic to specific limits depending on the policies. The core routers interpret the DS field according to the set of PHBs defined for them. Thus, the policy definition for core routers must specify the type of queuing behavior that corresponds to different packet markings [5].

3. The Implementation Model Architecture

3.1 Architecture

The outline of our implementation model for policy-based DiffServ QoS management system is shown in Figure 2.
The model roughly consists of a Web server and an EJB-based policy server. We use a Web browser-based interface as Graphical User Interface (GUI). To determine the topology of the DiffServ network, and to create and administer high-level QoS policies to be enforced on the DiffServ network, an administrator uses the Web browser-based interface of an EJB-based policy server. In order to create high-level QoS policies and convert them into low-level QoS policies, the routing topology of the DiffServ network must be known. The topology information required for enforcing policies is discovered by using two SNMP MIB-II tables, an ipAddrTable and an ipRouteTable. The topology information retrieved from the network is stored in a Topology Database and is represented as Topology Node (TN) entity beans in the EJB container. This procedure is guided by a Topology Management Agent (TMA) bean that is a session bean. A Java Database Connectivity (JDBC) driver connects the EJB container to the Topology Database. High-level DiffServ QoS policies are represented as valid XML files through a Java servlet and low-level ones are represented as policy entity beans in the EJB container of the EJB-based policy server through a Policy Management Agent (PMA) session bean. In our model, there are
three policy beans for managing QoS of the DiffServ network: Classifier (CL) policy, Meter & Action (MA) policy, and Queuing & Scheduling (QS) policy. The CL policy and MA policy are usually used in edge routers, while QS policy is usually used in core routers. They are stored in a QoS policy directory. The QoS management operations processing those policy beans are guided by a QoS Management Agent (QMA) session bean. The Java Naming and Directory Interface (JNDI) API and the Lightweight Directory Access Protocol (LDAP) provide lookup services for the QoS policy directory. Our current implementation model uses SNMP to distribute those low-level policy beans, namely, the EJB-based policy server converts the low-level policies into the appropriate DiffServ MIB [6] values that will conform to device appropriately. The QoS monitoring information of the DiffServ network is also obtained from the DiffServ MIB. We have established a set of DiffServ routers in the Linux systems and have added an SNMP agent for the DiffServ MIB and MIB-II in each router.

A Service Management Agent (SMA) session bean, the Service entity beans, and the Customer entity beans need for service management system. A SMA session bean provides access to other management beans and guides the service management on behalf of the client system. It creates customer accounts, provides service information, and guides the provisioning state model by communicating to internal management beans. The Customer bean represents the view of a customer. The individual Customer bean needs to access data across the tables of the Customer Service database to query for individual customer. The Service bean represents the class of service types that is associated with a service. An instance of a Service bean represents a service, which is provisioning for a customer [8, 9].

Our DiffServ QoS management system conforms to the Model-View-Controller (MVC) model and Session Façade pattern. And so it is highly manageable and scalable, and provides the overall strategy for the clear distribution of objects involved in managing service.

3.2 Topology and Role Discovery

In order to manage the QoS of a DiffServ network, we must obtain the routing topology and role information for all the DiffServ edge routers and core routers. Within the policy discipline of DiffServ networks, a router might have the role of an edge router, or it might have the role of a core router.

The routing topology information required for enforcing policies is discovered by using two SNMP MIB tables, an ipAddrTable and an ipRouteTable. By combining the two table entries we can obtain the routing topology information. The role information of routers also is discovered by using the ipAddrTable. All DiffServ edge routers will have at least one interface address in DiffServ domain and at least one interface with the customer’s address. In such an environment, if all interfaces of a router belong to same subnet, the router’s role is a core router. Otherwise the router’s role is an edge router. The topology and role information is stored in a Topology database and is represented as TN entity beans in the EJB container of the policy server. A set of these actions are guided by a TMA session bean.
The algorithm used to discover routing topology and role is based on the information stored in a router’s ipAddrTable and ipRouteTable. The ipAddrTable contains IP address of all network interfaces in a router and the ipRouteTable contains the IP routing table that has the next hop host and network interface for a set of destination IP addresses. Thus the direct neighbors of this host can be learned by retrieving the two tables. The following is the detailed algorithm.

```java
// set out the topology discovery from its own machine
startNode = DiffServ_QoS_management_system;
getTopology_Role(startNode);
....

// Retrieve the ipAddrTable and ipRouteTable of this host
void getTopology_Role(Host host) {
    getSNMP ipAddrTable from this host;
    Filter out all interfaces on each valid interface;
    Add all interfaces to this host’s interface;

    // set role
    if(all interfaces belong to same subnet)
        host.role = coreRouter;
    else
        host.role = edgyRouter;

    Get SNMP ipRouteTable from this host;
    Filter out all neighbors on each valid route;
    Add all neighbors to this host’s neighbor list;

    if(There aren’t any interface && direct_neighbor)
        return;

    // Iterate neighbors
    Host nextHost = host.getDirectNeighbor();
    while (nextHost is not NULL) {
        getTopology_Role(nextHost); // recursion
        nextHost = host.getNextDirectNeighbor();
    }
}
```

The topology and role discovery algorithm starts by looking at own machine’s two SNMP MIB tables. Therefore, the EJB-based policy server must have a local SNMP agent for MIB-II, and must have a direct connection with an edge router in the DiffServ network.

3.3 Policy Management

High-level DiffServ QoS policies are created as valid XML files by a Java servlet act as a front-end controller and low-level ones are represented as instances of the three policy entity beans through a session bean. There are three QoS policy entity beans for
managing QoS of the DiffServ network: CL policy entity bean, MA policy entity bean, and QS policy entity bean. The CL policy bean classifies packet flows and assigns Classifier Identities (CIDs) on them. A MA policy bean meters packets and performs actions on them: marker, counter, absolute dropper, and so forth. A QS policy bean queues and schedules, or drops packets. By using QoS policy entity beans, the PMA session bean provides the policy translation and the QMA session bean provides the policy distribution. Figure 3 shows the sequence diagram for a set of policy management procedures. As demonstrated in Figure 3, the policy management procedures composed of policy creation, policy validation, policy translation, and policy distribution.

3.3.1 Policy Validation and Translation
The most basic check of policy validation is that of validating the syntax of the policy specification. In our model we defined a Document Type Definition (DTD) file to validate high-level QoS policies described as XML files. A Java servlet receives the data from an administrator and creates XML policy files and then check the validation of them through a DTD. After a set of these actions, the Java servlet requests a PMA session bean instance that it creates entity bean instances of low-level QoS policies. The PMA session bean provides the policy translation through three QoS policy entity beans. Entity beans represent the behavior and data of a business object, while session beans model the workflow of entity beans and another session beans. In this model, the PMA bean uses a CL policy bean, a MA policy bean, a QS policy bean to translate high-level QoS policies to low-level QoS policies. High-level policies related to edge routers may be applied to CL policy beans and MA policy beans. While high-level policies related to core routers may be applied to QS policy beans. The policy management is accomplished a PMA session bean and a QMA session bean.
3.3.2 Policy Distribution Using SNMP
In our implementation model the DiffServ QoS policies mostly conform to the SNMP DiffServ MIB [6]. In other words, policy beans perform SNMP operations for DiffServ MIB. A QMA session bean provides the policy distribution through a CL policy bean, a MA policy bean, and a QS policy bean. A CL policy bean is usually deployed to the ingress interfaces of edge routers and applies to inbound traffic. MA policies are usually deployed to the egress interfaces of edge routers and apply to outbound traffic. A QS policy is usually deployed to the egress interfaces of core routers and applies to outbound traffic. A set of those actions are accomplished by using the DiffServ MIB. The DiffServ MIB describes the configuration and management aspects of DiffServ nodes. The MIB contains the functional elements of the datapath, using various tables. The idea is that RowPointers are used to combine the various functional elements into one datapath. The DiffServ MIB tables are categorized in four architectural DiffServ elements, which are classifier, meter, action, and queue. The main advantage of using SNMP for policy distribution is that it is likely to work across all the routers in a standard manner.

4. Implementation

4.1 DiffServ QoS Management System
Our DiffServ QoS management system is implemented in the Windows 2000 Server platform with Java. It includes both the presentation-tier and the business-tier. The presentation-tier runs a Web server to handle administrator requests, and invokes servlets and JSPs. We use Apache Tomcat 4.0.6 [13] for a servlet and JSP container in our application. An EJB-based policy server within the business-tier runs an EJB server to handle EJB components. We use JBoss 3.0.3 [14] for an EJB server and use EJB 1.1 to implement EJB beans. JBoss is an Open Source, standards-compliant, application server implemented in 100% Pure Java and distributed for free. AdventNet SNMP v2c APIs [15] written in Java are used for handling EJB-based policy server’s SNMP operations. The Oracle 8i Enterprise Edition for Windows 2000 is used for storing performance and topology information derived from MIB tables.

4.2 Linux-based DiffServ Networks
We have established a set of DiffServ routers in the Linux systems and have added an SNMP agent for the MIB-II and DiffServ MIB in each router. Support for DiffServ is already integrated into 2.4 kernels. In order to enable it, you may have to reconfigure and rebuild your kernel, or at least some modules [17]. A DiffServ agent has been implemented by using UCD-SNMP 4.2.5 [18] that provides the agent extension capability. Communication between the DiffServ agent and the Linux traffic control kernel use a netlink socket. Netlink is used to transfer information between kernel and...
user-space processes, over its bi-directional communications links. The DiffServ agent receives management operations from the low-level policy beans in the EJB-based policy server and performs the appropriate parameter changes in the Linux traffic control kernel. As the result of these operations, policies are distributed to PEPs that are Linux DiffServ routers and monitoring of these results is possible.

5. Conclusion

In his paper, we have proposed an implementation model of policy-based DiffServ QoS management system. The implementation model is designed according to multi-tier MVC model, IETF policy framework, and EJB technologies. In our model, high-level DiffServ QoS policies are represented as valid XML files through Java servlet and translated to low-level EJB policy beans in the EJB container of the DiffServ QoS management system through a PMA session bean. The model has three low-level EJB policy beans follow as CL policy, MA policy, and QS policy. CL policies and MA policies are deployed on edge routers, and QS policies are deployed on core routers. These Policies are distributed using SNMP to control and monitor QoS of DiffServ networks through a QMA session bean. To support this work, we implemented the SNMP DiffServ agent containing DiffServ MIB on the Linux-based DiffServ routers.

We are currently at the stage of implementation of the business logic on the EJB-based policy server. We plan to experiment with and demonstrate the system on laboratory testbeds using Linux-based router. Currently we are only checking the syntax validation of policy. So, we will also develop a policy conflict detection and resolution algorithm and apply that to our system. We intend to report result on detailed aspects of the proposed implementation model in future papers.

References